

UNITED STATES PATENT AND TRADEMARK OFFICE

---

BEFORE THE BOARD OF PATENT APPEALS  
AND INTERFERENCES

---

*Ex parte* Jeremy Alan Arnold and John Matthew Santosuosso

---

Appeal No. \_\_\_\_\_

Application No. 09/998,511

---

AMENDED APPEAL BRIEF

---

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicant: Jeremy Alan Arnold et al.                      Art Unit: 2192  
Serial No.: 09/998,511    Examiner: Chrystine Pham  
Filed: November 30, 2001  
For: INHERITANCE BREAKPOINTS FOR USE IN DEBUGGING  
OBJECT-ORIENTED COMPUTER PROGRAMS

---

Mail Stop Appeal Brief - Patents  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

**AMENDED APPEAL BRIEF**

**I. REAL PARTY IN INTEREST**

This application is assigned to International Business Machines Corporation, of Armonk, New York.

**II. RELATED APPEALS AND INTERFERENCES**

There are no related appeals or interferences.

**III. STATUS OF CLAIMS**

Claims 1-5, 7-15 and 17-30 are pending in the Application, and all are now on appeal.

**IV. STATUS OF AMENDMENTS**

No amendments have been filed subsequent to final rejection (Paper mailed May 4, 2005).

## V. SUMMARY OF CLAIMED SUBJECT MATTER

The invention is generally directed to utilization of an "inheritance breakpoint" to assist in debugging an object-oriented computer program having a method identified in a base class or interface and implemented by at least one method implementation in another class. (Application, p. 5). Creation of an inheritance breakpoint for a particular method responsive to a user request typically results in the automated generation of a breakpoint for all or a subset of the implementations of that method, often freeing a user from having to manually determine what particular implementation(s) implement that method. In many environments, the automated generation of implementation breakpoints may be performed during initial loading of classes within which such implementations may be defined. (Application, p. 5).

Breakpoints have traditionally been used by computer developers when debugging computer programs to specify particular instructions to halt execution of such programs during debugging, e.g., to enable a developer to display the current state of a program for the purpose of diagnosing any bugs or performance problems in the program. (Application, p. 2). However, conventional breakpoints often have limited utility when debugging object-oriented programs.

Object-oriented programs are typically created using templates referred to as "classes", which associate a collection of declared data with a set of operations or methods capable of being performed on that data. (Application, p. 3). A fundamental feature of object-oriented programming is that classes can be related to one another by inheritance. The properties, behaviors, data and operations of a parent class, i.e., a "base class", may be inherited without modification by some child class, or "subclass", or the behavior, properties and operations may be selectively refined under the control of the programmer in the dependent class. (Application, p. 3). Moreover, in some object-oriented programming environments, a developer is permitted to separate a public "interface" for a class from the specifics of its implementation and/or to provide "abstract" classes, which themselves cannot be instantiated, but which can serve as templates for any subclasses so that any method and data incorporated in the abstract classes will be inherited within the subclasses. (Application, p. 3).

As described in the Application, an "implementation" of a method that is identified in a particular class can be found in that class or in one of its subclasses. Likewise, if a method is

identified in an interface, the implementation of that method may be found in a class that implements the interface, and if a method is identified in an abstract class, the implementation of that method may be found in a non-abstract class that implements the abstract class.

(Application, p. 8). Regardless of wherever a method identified in a class, abstract class or interface is implemented, the name, or identifier, of that method typically remains the same, and as a result, whenever a program makes a call to a particular method, it may not be readily apparent to a programmer exactly what implementation of that method will be executed as a result of the call. (Application, p. 4). Therefore, in order to debug that section of a program, a programmer conventionally was required to hunt through the program to locate all of the possible implementations of a method and manually set breakpoints on all of those implementations. Doing so, however, can be exceptionally time consuming (Application, p. 4).

The claims on appeal address these problems through the use of "inheritance" breakpoints that halt the execution of a computer program whenever an implementation of a method is reached during execution of the computer program. With regard to independent claims 1, 13, 14, 15, 18, 26, 28 and 30, an inheritance breakpoint is associated with a particular program entity in a computer program that identifies a method, but is capable of halting execution of the computer program in response to reaching an implementation of that method that is defined in a program entity other than that with which the inheritance breakpoint is specifically associated.

(Application, p. 5). As such, by setting an inheritance breakpoint on a particular method in a class, a developer need not seek out any other implementations of the method in any dependent classes to ensure that breakpoints are set on all possible implementations of the method, thus saving the developer the time and effort associated with finding such implementations, and avoiding the possibility that a developer that would otherwise be required to manually search for all possible implementations might miss one of the implementations.

With respect to claims 1, 15, 18, 26, 28 and 30, the other entity depends from the entity with which the inheritance breakpoint is specifically associated, whereby the implementation of a method may be in a subclass, an implementation of an interface or an implementation of an abstract class within which an inheritance breakpoint is defined. (Application, p. 8). For claims 2, 14 and 19, the other entity depends from the entity with which the inheritance breakpoint is

specifically associated is related as an implementation of an interface, while for claims 5 and 21, the entities are related as abstract class and implementation of the abstract class. Claims 8, 9, 15, 23, 24, 26 and 30 additionally recite the concept of implementing an inheritance breakpoint associated with a particular method by setting breakpoints on at least a subset of the implementations of the method in any dependent classes. (Application, pp. 5-6).

Using the aforementioned inheritance breakpoints, a programmer is permitted, for example, to set an inheritance breakpoint for a particular method, and then have that inheritance breakpoint trigger a halting of execution whenever any of the implementations of that method are reached during execution of the computer program. As such, a programmer may be relieved of the burden of hunting through lengthy program code to locate each implementation of a method whenever a programmer is unsure as to which implementation of a method will be executed in response to a given method call. (Application, p. 5).

## VI. GROUND S OF REJECTION TO BE REVIEWED ON APPEAL

- A. Claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24 and 26-30 stand rejected under 35 U.S.C. § 102(b) as being anticipated by U.S. Patent No. 5,845,125 to Nishimura et al. (hereinafter *Nishimura*).
- B. Claims 2, 5, 14, 19 and 21 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Nishimura* and U.S. Patent No. 5,093,914 to Coplien et al. (hereinafter *Coplien*).
- C. Claim 11 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Nishimura* in further view of U.S. Patent No. 5,740,440 to West (hereinafter *West*).
- D. Claims 12 and 25 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Nishimura* in view of Applicant's Admission of Prior Art (hereinafter *AAPA*).

## VII. ARGUMENT

Applicants respectfully submit that the Examiner's rejections of claims 1-5, 7-15 and 17-30 are not supported on the record, and should be reversed.

A. Claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24 and 26-30 were improperly rejected as being anticipated by *Nishimura*.

The Examiner argues that *Nishimura* anticipates all of claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24 and 26-30. Anticipation of a claim under 35 U.S.C. §102, however, requires that "each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros., Inc. v. Union Oil Co.*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987), *quoted in* *In re Robertson*, 49 USPQ2d 1949, 1950 (Fed. Cir. 1999). Absent express description, anticipation under inherency requires extrinsic evidence that makes it clear that "the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill." *Continental Can Co. v. Monsanto Co.*, 20 USPQ2d 1746, 1749 (Fed. Cir. 1991), *quoted in* *In re Robertson* at 1951. "Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient." *Continental Can* at 1749, *quoted in* *In re Robertson* at 1951.

Applicants respectfully submit that *Nishimura* does not disclose the various features recited in claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24 and 26-30, and as such, the rejections thereof should be reversed. Applicants will hereinafter address the various claims that are the subject of the Examiner's rejection in order.

### *Independent Claim 1*

Independent claim 1 generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes, in response to user input, setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity. The method also includes halting execution of the object-oriented computer program during debugging in response to

reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity.

As such, it is important to note that execution is halted in response to hitting an implementation of a method in a program entity that “depends” from another program entity within which a breakpoint has been set. By doing so, a user is able to specify a given method to associate with a breakpoint, and then have a debugger halt execution whenever any implementation of that method is hit.

Claim 1 has been rejected as being anticipated by *Nishimura*, which discloses a debugger for an object-oriented environment. However, *Nishimura* is specifically directed to setting breakpoints on individual objects, or instances, of a class, rather than upon all instances of a class. Furthermore, *Nishimura* is directed to addressing the problem that arises due to inheritance, whereby a given instance of a class that inherits from a parent or base class may have certain methods and data that are defined in the base class, rather than the specific class for the instance, which for prior designs required a user to set individual breakpoints in a class and any of its base classes to ensure that a debugger will break on a desired object. (col. 4, lines 9-33).

*Nishimura* addresses this problem through the use of “object-type” breakpoints, which results in the automated setting of breakpoints in all of the public functions of a class and of any base (parent) classes therefor (col. 15, lines 11-13, and Fig. 9). Of note, this mechanism in *Nishimura* for automatically setting breakpoints in the base or parent classes for a given object so that a program will be halted when a breakpoint in a base class is encountered is precisely the opposite scenario to that recited in claim 1. Specifically, claim 1 recites halting execution at an implementation of a method that is defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated, while *Nishimura* sets breakpoints in parent or base classes of a given object.

By way of example, consider a program including three classes, A, B and C, where class B is a subclass of class A, and class C is a subclass of class B. Consider also that class B has a method X, and class C overrides this method with a method X’. Using the invention recited in claim 1, a user may be permitted to set an inheritance breakpoint on method X of class B, and have the program halted upon encountering method X’ in class C. Conversely, in the *Nishimura*

system, a user may be permitted to set a breakpoint on an instance of class B, and have a breakpoint automatically set in class A. Given that class C depends from class B, however, *Nishimura* would not automatically set any breakpoint on an instance of class C as a result of setting the breakpoint on the instance of class B.

In the final Office Action dated May 4, 2005, the Examiner attempts to rebut Applicants' arguments by citing col. 2, line 8 to col. 4, line 40 of *Nishimura*. However, col. 2, lines 8-60 merely discloses the general concept of inheritance, as well as identifies the relationships between base and child classes. Col. 2, line 61 to col. 3, line 28 generally discloses the difficulties encountered with respect to the display of execution locations, the setting of breakpoints, and the logging of data, based upon the fact that conventional debuggers are based upon specific locations in a program, rather than in specific objects.

Col. 3, line 29 to col. 4, line 40 discusses the concept of inheritance, but contrary to the Examiner's assertions, fully supports Applicants position with regard to patentability over *Nishimura*. In particular, the Examiner appears to place significant weight on the disclosure at col. 4, lines 9-20, which describes an example where a class [character-string] is a subclass or child class of a base or parent class [array], such that the subclass [character-string] includes an operation that is unique to the subclass, as well as another operation that is inherited from the base or parent class [array]. *Nishimura* specifically states that, if a user wants to halt execution on a particular object a that is of the type [character string], "the user must set a breakpoint not only in the class [character-string] operation but also in the operation inherited from the class [array]." Put another way, *Nishimura* suggests to the user the desirability of setting a breakpoint in a parent or base class of a particular object, but speaks nothing about setting breakpoints in subclasses of a particular object. It should also be noted that the operations being disclosed in this passage of *Nishimura* are manual operations where individual breakpoints are set, which falls far short of disclosing the features recited in claim 1.

*Nishimura* also focuses on the desirability of setting breakpoints on base classes for a particular object in the passage that continues at col. 4, lines 21-33:

When the user sets a breakpoint in all the public functions in the class of the object, he must set a breakpoint in all the public functions that exist in the class of the object and, at the same time, in all the public functions that exist in indirect base classes if the class the user wants to break has inherited some functions from other classes. (*emphasis added*).

*Nishimura* does mention, at col. 4, lines 34-40, that when execution of an object stops as a result of a breakpoint, the user may have difficulty determining what kind of object was hit, be it the object of interest, an object generated from an indirect base class, an object generated from another class that inherits from an indirect base class, or an object generated from a descendent class of one of those classes. This disclosure, however, does not teach the concept of halting execution of a program in response to hitting an implementation of a method in one entity that is dependent upon another entity within which a breakpoint is set. Rather, this disclosure simply acknowledges that when a breakpoint is set in a base class of a particular object, when that breakpoint is hit, the object that hit the breakpoint might be the object of interest, or might be another object that inherits from the same base class. As an example, if there is a base class A, two child classes B and C that inherit from class A, a grandchild class D that inherits from class B, and a grandchild class E that inherits from class C, if a breakpoint is set on an operation in class A, the breakpoint will be hit any time that operation is called for any objects created from any of classes A-E.

The Examiner also apparently relies on col. 14, line 55 to col. 15, line 25 of *Nishimura*; however, as discussed above, this passage does not disclose the concept of halting execution of a program in response to hitting an implementation of a method in one entity that is dependent upon another entity within which a breakpoint is set. To the contrary, as Applicants have noted above, this passage states explicitly that "[w]hen the user selects an object-type breakpoint, the breakpoint setting section 16 sets a breakpoint in all the public functions of the class and of the indirect base classes." (col. 15, lines 11-13). Thus, this passage of *Nishimura* does not support

the Examiner's position, and as Applicants have noted above, discloses precisely the opposite configuration to that recited in claim 1.

Given, therefore, that *Nishimura* discloses the opposite configuration to claim 1, *Nishimura* does not disclose halting execution in response to reaching an implementation of a method defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated, as required by claim 1. Claim 1 is therefore novel over *Nishimura*, and the rejection thereof should be reversed.

Claim 1 is also non-obvious over *Nishimura*, as the reference is directed to solving a completely different problem, and as the solution proposed by *Nishimura* results in precisely the opposite configuration to that recited in claim 1. The invention recited in claim 1 is directed to halting execution of a program based upon a particular method, regardless of within which entity among a chain of entities that method is implemented, and furthermore, irrespective of which instance of the entity hits the method. *Nishimura*, on the other hand, is directed to halting execution of a program based upon a particular instance of an object, rather than halting on every instance of the object. The invention of claim 1 looks downward towards subclasses and implementations of interfaces and abstract classes to halt execution, while *Nishimura* looks upward toward base classes.

The Examiner has therefore failed to establish anticipation of claim 1 by *Nishimura*. Reversal of the Examiner's rejection of claim 1 is therefore respectfully requested. Furthermore, in view of the non-obviousness of claim 1 over *Nishimura*, allowance of claim 1, and of claims 2-5 and 7-12 which depend therefrom, are respectfully requested.

#### Dependent Claims 3-4 and 7

Claims 3-4 and 7 are not argued separately.

#### Dependent Claim 8

Claim 8 depends from claim 1, and additionally recites the step of, during loading of a class in the object-oriented computer program, identifying each implementation of the method in the class and setting a breakpoint on such implementation. Claim 8 also recites that halting

execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation.

In rejecting claim 8, the Examiner relies on Fig. 9, col. 4, lines 21-33, col. 15, lines 11-14 and col. 15, lines 23-26. However, in each of these passages it is quite evident that *Nishimura* discloses the concept of identifying those methods that are inherited from base classes and setting breakpoints on those methods -- once again the precisely opposite configuration to that recited in claim 8. Claim 8 recites identifying "implementations" of a method in a class, which are described in the Application as filed as being methods defined in subclasses, methods defined in classes that implement interfaces, or methods defined in classes that implement abstract classes (*see, e.g.*, Application p. 8, lines 18-21). An "implementation" of a method within the context of the invention is not found in a base class, and as such, the cited passages in *Nishimura* simply do not disclose the additional features recited in claim 8. Reversal of the Examiner's rejection of claim 8 is therefore respectfully requested.

#### Dependent Claim 9

Similar to claim 8, claim 9 recites the step of setting a breakpoint on each implementation of the method, and that halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation.

In rejecting claim 9, the Examiner relies the arguments related to claim 8, which relied on Fig. 9, col. 4, lines 21-33, col. 15, lines 11-14 and col. 15, lines 23-26. However, as discussed above in connection with claim 8, it is quite evident from each of these passages that *Nishimura* discloses the concept of setting breakpoints on methods that are identified in base classes -- once again the precisely opposite configuration to that recited in claim 9. Claim 9 recites setting breakpoints on "implementations" of a method in a class, which are described in the Application as filed as being methods defined in subclasses, methods defined in classes that implement interfaces, or methods defined in classes that implement abstract classes (*see, e.g.*, Application p. 8, lines 18-21). An "implementation" of a method within the context of the invention is not found in a base class, and as such, the cited passages in *Nishimura* simply do not disclose the

additional features recited in claim 9. Reversal of the Examiner's rejection of claim 9 is therefore respectfully requested.

Dependent Claim 10

Claim 10 is not argued separately.

Independent Claim 13

Next with respect to independent claim 13, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes setting an inheritance breakpoint that is associated with a first class in the object-oriented computer program in which is identified a method in response to user input, and halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second class in the object-oriented computer program that inherits from the first class.

Claim 13 is rejected as being anticipated by *Nishimura*. However, as discussed above in connection with claim 1, *Nishimura* discloses automatically setting breakpoints in parent or base classes of a given class instance. Claim 13, however, discloses an opposite scenario where execution is halted in response to reaching an implementation of a method in a class that inherits from another class with which an inheritance breakpoint is associated. Applicants therefore respectfully submit that claim 13 is novel over *Nishimura*, and the Examiner's rejection thereof should be reversed. Furthermore, claim 13 is non-obvious over the reference as there is no suggestion in *Nishimura* of the desirability of halting execution in a subclass of a particular class with which a breakpoint is associated, as discussed above in connection with claim 1. The rejection of claim 13 should therefore be reversed.

### Independent Claim 15

Next with respect to independent claim 15, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes receiving user input to halt program execution during debugging in response to reaching any of a plurality of implementations of a method in an object-oriented computer program, and thereafter setting a breakpoint for at least a subset of the plurality of implementations such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set. Claim 15 also recites that the user input to halt program execution includes user input to set an inheritance breakpoint on the method, that the inheritance breakpoint is associated with a first program entity, and that at least one of the plurality of implementations of the method is defined in a second program entity that depends from the first program entity. The claim also recites that the subset of implementations upon which a breakpoint is set includes the implementation defined in the second program entity.

As discussed above in connection with claim 1, *Nishimura* discloses only the setting of breakpoints in base or parent classes of a given instance of a class. As claim 15 recites automatically setting a breakpoint in a program entity that depends from another program entity (e.g., a subclass or an implementation of an interface), claim 15 is an opposite scenario to that disclosed in *Nishimura*. It is also important to note that the setting of the breakpoint in the entity that depends from the other entity is in response to user input to set an inheritance breakpoint on the method in the other entity, so the Examiner could not argue that claim 15 reads on any manual setting of breakpoints in multiples implementations of a method, regardless of the inheritance relationship between those implementations of the method. Claim 15 is therefore novel and non-obvious over *Nishimura*, and the rejection of claim 15 should be reversed.

### Dependent Claim 17

Claim 17 is not argued separately.

### Independent Claims 18 and 28

Next with respect to independent claims 18 and 28, each of these claims recites *inter alia* the concept of halting execution upon reaching an implementation of a method defined in a second program entity that depends from a first program entity with which is associated an inheritance breakpoint. As discussed above in connection with claim 1, this combination of features is not disclosed or suggested by *Nishimura*. The rejections of independent claims 18 and 28 should therefore be reversed for the same reasons presented above in connection with claim 1.

### Dependent Claims 19-22

Claims 19-22 are not argued separately.

### Dependent Claims 23-24

Claims 23-24 each recite *inter alia* the concept of setting a breakpoint on each implementation of a method. As described above in connection with claims 8 and 9, *Nishimura* discloses the concept of setting breakpoints on methods that are identified in base classes, while these claims focus on setting breakpoints on "implementations" of a method in a class, which are described in the Application as filed as being methods defined in subclasses, methods defined in classes that implement interfaces, or methods defined in classes that implement abstract classes. An "implementation" of a method within the context of the invention is not found in a base class, and as such, the cited passages in *Nishimura* simply do not disclose the additional features recited in claim 23-24. Reversal of the Examiner's rejections of claim 23-24 is therefore respectfully requested.

### Independent Claims 26 and 30

Likewise, with respect to independent claims 26 and 30, each of these recite the concept of setting a breakpoint on an implementation of a method in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated. Claims 26 and 30 are therefore novel and non-obvious over the prior art of record for the same reasons presented above for claim 15. The rejections of claims 26 and 30 should be reversed.

Dependent Claims 27 and 29

Claim 27 and 29 are not argued separately.

B. Claims 2, 5, 14, 19 and 21 were improperly rejected as being unpatentable over *Nishimura* and *Coplien*.

The Examiner argues that claims 2, 5, 14, 19 and 21 are obvious over *Nishimura* and *Coplien*. However, a *prima facie* showing of obviousness requires that the Examiner establish that the differences between a claimed invention and the prior art "are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. §103(a). Such a showing requires that all claimed features be disclosed or suggested by the prior art, along with objective evidence of the suggestion, teaching or motivation to combine or modify prior art references, as "[c]ombining prior art references without evidence of such a suggestion, teaching or motivation simply takes the inventor's disclosure as a blueprint for piecing together the prior art to defeat patentability -- the essence of hindsight." In re Dembiczak, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999).

Among claims 2, 5, 14, 19 and 21, only claim 14 is independent. Claim 14 recites a computer-implemented method of debugging an object-oriented computer program, which includes setting an inheritance breakpoint that is associated with an interface in the object-oriented computer program in which is identified a method in response to user input, and halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a class in the object-oriented computer program that implements the interface.

Claims 2 and 19 depend from claims 1 and 18 and recite that the first program entity is an interface that identifies the method, and that the second program entity is a class that implements the method. Similarly, claims 5 and 21 depend from claims 1 and 18 and recite that the first program entity is an abstract class that identifies the method, and that the second program entity is a non-abstract class that implements the method.

In rejecting each of these claims based upon *Nishimura* and *Coplien*, the Examiner has acknowledged that *Nishimura* does not disclose the concept of setting a breakpoint that is

associated with an interface or an abstract class and halting execution upon reaching an implementation of a method defined in a class that implements the interface. Instead, the Examiner relies on *Coplien* for allegedly disclosing an interface/implementation relationship in object-oriented programming. However, *Coplien* merely discloses the general concept of an interface-implementation relationship in object-oriented programming, but falls short of disclosing halting execution of a program upon reaching an implementation of a method identified in an interface with which a breakpoint is associated. What *Coplien* does disclose is a technique similar to *Nishimura*, whereby a breakpoint is set on a specific instance of an object, when the address of a function defined in the instance is not known until runtime. Of note, however, *Coplien* does not disclose or suggest setting a breakpoint on an implementation of a method in an interface or an abstract class.

Given that *Nishimura* discloses setting breakpoints on base or parent classes, rather than in dependent entities such as subclasses, implementations of interfaces, implementations of abstract classes, etc., Applicants submit that one of ordinary skill in the art would not be motivated by *Coplien* to modify *Nishimura* to set breakpoints on dependent entities such as interface implementations or implementations of abstract classes. Put another way, neither *Nishimura* nor *Coplien* discloses or suggests the concept of setting a breakpoint on a first entity, be it an interface or an abstract class, coupled with halting execution in response to hitting an implementation of a method in a class that implements an interface or an abstract class.

In the final Office Action dated May 4, 2005, the Examiner attempts to rebut Applicants' arguments by characterizing Applicants' arguments as merely attacking the references individually. However, there must be some objective evidence presented of a motivation to modify *Nishimura* to halt execution of a program in response to hitting an implementation of a method defined in an implementation class for an interface. As noted above, however, since neither reference discloses or suggests the concept of halting a program in response to hitting an implementation of a method in *any* dependent entity, much less specifically within an implementation of an interface, the Examiner has failed to provide sufficient evidence to sustain a *prima facie* case of obviousness.

Consequently, Applicants submit that the Examiner has failed to establish a *prima facie* case of obviousness as to any of claims 2, 5, 14, 19 and 21. The rejections of each of these claims should therefore be reversed.

C. Claim 11 was improperly rejected as being unpatentable over *Nishimura* in further view of *West*.

Claim 11 depends from claim 9, and therefore incorporates the limitations of that claim. Accordingly, Applicants submit that claim 11 is patentable over *Nishimura* for the same reasons as presented above for claims 1 and 9. Moreover, *West* adds nothing to the Examiner's rejection in this regard, as the reference is cited only for its alleged teaching related to setting breakpoints on method calls. Reversal of the Examiner's rejection of claim 11, and allowance of the claim, are therefore respectfully requested.

D. Claims 12 and 25 were improperly rejected as being unpatentable over *Nishimura* in view of *AAPA*.

Claim 12 depends from claim 1, while claim 25 depends from claim 18, and therefore each claim incorporates the limitations of its respective base claim. Accordingly, Applicants submit that claims 12 and 25 are patentable over *Nishimura* for the same reasons as presented above for claims 1 and 18. Moreover, *AAPA* adds nothing to the Examiner's rejection in this regard, as the Examiner is only asserting that the concept of a conditional breakpoint in the abstract is known in the art. Reversal of the Examiner's rejections of claims 12 and 25, and allowance of these claims, are therefore respectfully requested.

### CONCLUSION

In conclusion, Applicants respectfully request that the Board reverse the Examiner's rejections of claims 1-5, 7-15 and 17-30, and that the Application be passed to issue. If there are any questions regarding the foregoing, please contact the undersigned at 513/241-2324.

Moreover, if any other charges or credits are necessary to complete this communication, please apply them to Deposit Account 23-3000.

Respectfully submitted,

WOOD, HERRON & EVANS, L.L.P.

Date: September 11, 2006

2700 Carew Tower  
441 Vine Street  
Cincinnati, Ohio 45202  
(513) 241-2324

By: /Scott A. Stinebruner/  
Scott A. Stinebruner  
Reg. No. 38,323

**VIII. CLAIMS APPENDIX: CLAIMS ON APPEAL (S/N 09/998,511)**

1. (Once Amended) A computer-implemented method of debugging an object-oriented computer program, the method comprising:

(a) in response to user input, setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity; and

(b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity.

2. (Original) The computer-implemented method of claim 1, wherein the first program entity is an interface that identifies the method, and wherein the second program entity is a class that implements the method.

3. (Original) The computer-implemented method of claim 1, wherein the first program entity is a first class that includes a second implementation of the method, wherein the second program entity is a second class that inherits from the first class, and wherein the first implementation of the method in the second class overrides the second implementation of the method in the first class.

4. (Original) The computer-implemented method of claim 3, wherein the second class is a subclass of the first class.

5. (Original) The computer-implemented method of claim 1, wherein the first program entity is an abstract class that identifies the method, and wherein the second program entity is a non-abstract class that implements the method.

6. (Canceled).

7. (Once Amended) The computer-implemented method of claim 1, wherein setting the inheritance breakpoint includes storing in a breakpoint data structure an entry that identifies the first program entity and the method.

8. (Original) The computer-implemented method of claim 1, further comprising, during loading of a class in the object-oriented computer program, identifying each implementation of the method in the class and setting a breakpoint on such implementation, wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation.

9. (Original) The computer-implemented method of claim 1, further comprising setting a breakpoint on each implementation of the method, wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation.

10. (Original) The computer-implemented method of claim 9, wherein setting a breakpoint on each implementation of the method includes setting a breakpoint on a first statement in an implementation of the method.

11. (Original) The computer-implemented method of claim 9, wherein setting a breakpoint on each implementation of the method includes setting a breakpoint on a method call to an implementation of the method.

12. (Original) The computer-implemented method of claim 1, wherein setting the inheritance breakpoint includes associating a user-specified condition with the inheritance breakpoint, and wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method is performed only if the user-specified condition has been met.

13. (Original) A computer-implemented method of debugging an object-oriented computer program, the method comprising:

(a) in response to user input, setting an inheritance breakpoint that is associated with a first class in the object-oriented computer program in which is identified a method; and

(b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second class in the object-oriented computer program that inherits from the first class.

14. (Original) A computer-implemented method of debugging an object-oriented computer program, the method comprising:

(a) in response to user input, setting an inheritance breakpoint that is associated with an interface in the object-oriented computer program in which is identified a method; and

(b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a class in the object-oriented computer program that implements the interface.

15. (Once Amended) A computer-implemented method of debugging an object-oriented computer program, the method comprising:

(a) receiving user input to halt program execution during debugging in response to reaching any of a plurality of implementations of a method in an object-oriented computer program, wherein the user input to halt program execution includes user input to set an inheritance breakpoint on the method, wherein the inheritance breakpoint is associated with a first program entity, and wherein at least one of the plurality of implementations of the method is defined in a second program entity that depends from the first program entity; and

(b) thereafter setting a breakpoint for at least a subset of the plurality of implementations including the implementation defined in the second program entity such

that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set.

16. (Canceled).

17. (Original) The computer-implemented method of claim 15, wherein setting the breakpoint includes, during loading of a class in the object-oriented computer program, identifying each implementation of the method in the class and setting a breakpoint on such implementation.

18. (Once Amended) An apparatus, comprising:

(a) a memory within which is resident at least a portion of an object-oriented computer program under debug, the object-oriented computer program including a first program entity in which is identified a method, and a second program entity that is different from and that depends from the first program entity, and that includes an implementation of the method; and

(b) program code configured to set an inheritance breakpoint that is associated with the first program entity and with the method in response to user input, and to halt execution of the object-oriented computer program during debugging in response to reaching the implementation of the method defined in the second program entity.

19. (Original) The apparatus of claim 18, wherein the first program entity is an interface that identifies the method, and wherein the second program entity is a class that implements the method.

20. (Original) The apparatus of claim 18, wherein the first program entity is a first class that includes a second implementation of the method, wherein the second program entity is a second class that inherits from the first class, and wherein the first implementation of the method in the second class overrides the second implementation of the method in the first class.

21. (Original) The apparatus of claim 18, wherein the first program entity is an abstract class that identifies the method, and wherein the second program entity is a non-abstract class that implements the method.

22. (Original) The apparatus of claim 18, wherein the inheritance breakpoint is additionally associated with the method, and wherein the program code is configured to store in a breakpoint data structure an entry that identifies the first program entity and the method.

23. (Original) The apparatus of claim 18, wherein the program code is further configured to set a breakpoint on each implementation of the method, and wherein the program code is configured to halt execution of the object-oriented computer program during debugging in response to reaching a breakpoint set on such implementation.

24. (Original) The apparatus of claim 23, wherein the program code is configured to set the breakpoint on each implementation of the method by dynamically setting a breakpoint on each implementation of the method in a class in the object-oriented computer program during loading of the class.

25. (Original) The apparatus of claim 18, wherein the program code is configured to associate a user-specified condition with the inheritance breakpoint, and wherein the program code is configured to halt execution of the object-oriented computer program during debugging in response to reaching the implementation of the method only if the user-specified condition has been met.

26. (Once Amended) An apparatus, comprising:

(a) a memory within which is resident at least a portion of an object-oriented computer program under debug, the object-oriented computer program including a method and a plurality of implementations of the method, wherein the method is identified in a first program entity, and wherein at least one of the plurality of

implementations of the method is defined in a second program entity that depends from the first program entity; and

(b) program code configured to receive user input to halt program execution during debugging in response to reaching any of the plurality of implementations of the method, and to thereafter set a breakpoint for at least a subset of the plurality of implementations such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set, wherein the user input to halt program execution includes user input to set an inheritance breakpoint on the method, wherein the inheritance breakpoint is associated with the first program entity, and wherein the program code is configured to set a breakpoint for the implementation defined in the second program entity.

27. (Once Amended) The apparatus of claim 26, wherein the program code is configured to set a breakpoint by, during loading of a class in the object-oriented computer program, identifying each implementation of the method in the class and setting a breakpoint on such implementation.

28. (Once Amended) A program product, comprising:

(a) program code configured to set an inheritance breakpoint in response to user input, wherein the inheritance breakpoint is associated with a first program entity in an object-oriented computer program and a method identified in the first program entity, and to halt execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity; and

(b) a signal bearing medium bearing the program code.

29. (Original) The program product of claim 28, wherein the signal bearing medium includes at least one of a transmission medium and a recordable medium.

30. (Once Amended) A program product, comprising:

(a) program code configured to receive user input to halt program execution of an object-oriented computer program during debugging in response to reaching any of a plurality of implementations of a method in the object-oriented computer program, and to thereafter set a breakpoint for at least a subset of the plurality of implementations such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set, wherein the method is identified in a first program entity, and wherein at least one of the plurality of implementations of the method is defined in a second program entity that depends from the first program entity, wherein the user input to halt program execution includes user input to set an inheritance breakpoint on the method, wherein the inheritance breakpoint is associated with the first program entity, and wherein the program code is configured to set a breakpoint for the implementation defined in the second program entity in connection with setting the breakpoint for the subset of the plurality of implementations; and

(b) a signal bearing medium bearing the program code.

**IX. EVIDENCE APPENDIX**

09/998,511

None.

**X. RELATED PROCEEDINGS APPENDIX**

09/998,511

None.